Detector Simulation

esipap

European School of Instrumentation
in Particle & Astroparticle Physics

Anna Zaborowska
CERN

7-8.02.2022

Acknowledgements

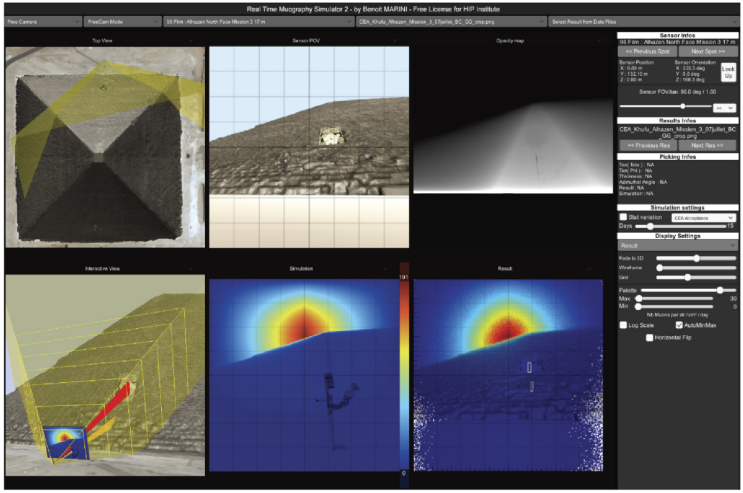
Based on previous lectures:

- Detector Simulation on ESIPAP 2021 by Alberto Ribon and Witek Pokorski
- Getting Started with Geant4 course by Mihaly Novak
- Monte Carlo Techniques by Bryan Webber

Detectors

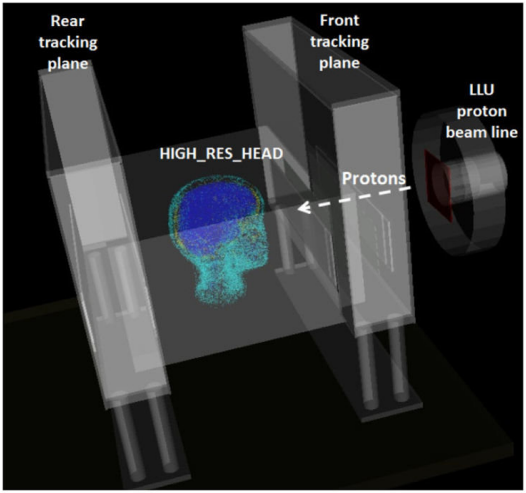
CMS-PHO-GEN-2017-009-6

Top view Sensor point of view Opacity map



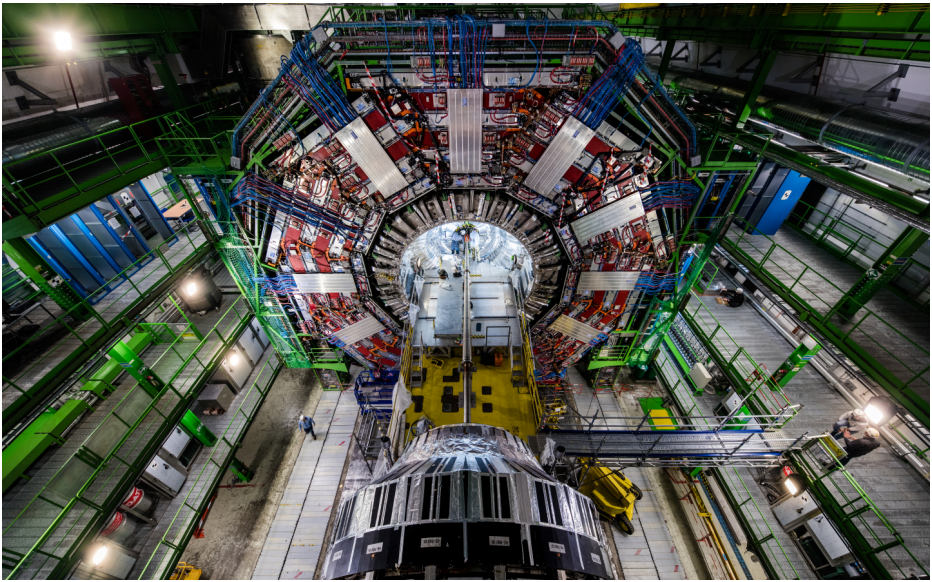
Interactive view Simulation Result

Nature volume 552, pages 386–390 (2017)



Phys. Med. 33 (2017)

XMM Newton X-Ray telescope



CERN-PHOTO-201703-062-52

Simulation

Why do we need simulation?

- Before building the detector - to design it;
- To operate the detector - background simulation, ...;
- For data analysis - to understand known and new phenomena;

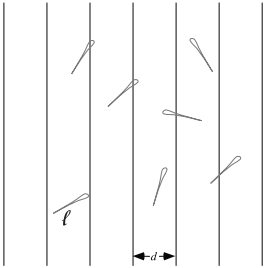
What is simulation?

A “virtual” experiment.

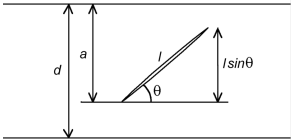
- Take known physics;
- Start from initial conditions (particles, materials, ...);
- Calculate final conditions;

Analytically? ... No!

Buffon's needle



Wolfram



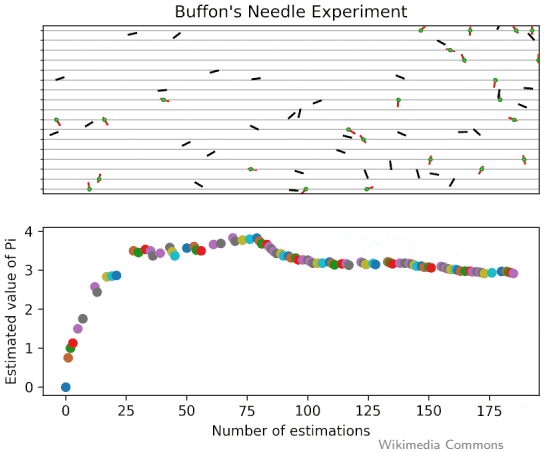
G. Holton, Value-at-Risk

$a \leqslant l \sin\theta$

- One of the oldest problems in the field of geometrical probability, first stated in 1777.
- Drop a needle on a lined sheet of paper and determine the probability of the needle crossing one of the lines.
- For $l < d$:

$$P = \int_{\theta=0}^{pi} \int_{a=0}^{l \sin\theta} \frac{1}{d\pi}\, da\, d\theta = \frac{2l}{d\pi}$$

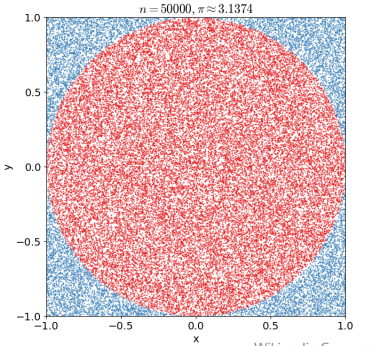
Buffon's needle



Probability P is directly related to the value of π.

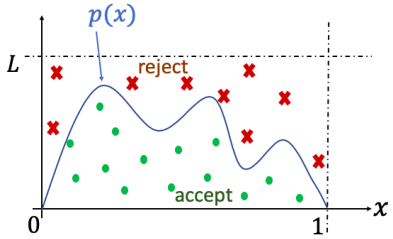
pi estimation

- Described by Laplace in 1886.
- Take circle of radius $r = 1$ inscribed in a square.
- Probability of random points falling inside the circle: $P = \frac{\pi}{4}$
- How to estimate π?
  - Draw uniformly N random points (x, y) from $(-1, 1)$ range.
  - Count the C points for which $x^2 + y^2 < 1$.
  - The ratio $\frac{C}{N}$ converges towards $\pi/4$.



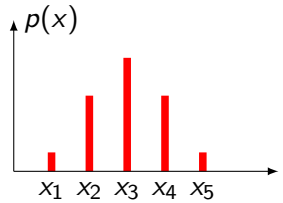
Wikimedia Commons

- Can be easily extended to any distribution



Random (stochastic) processes are widely used as mathematical models of phenomena that appear to vary in a random manner.

- Result cannot be specified in advance of observing it.
- Probabilities are used to describe the process.
- Discrete processes:
  - throwing a dice ⚀ ⚁ ⚂ ⚃;
  - selecting a decay channel for an unstable particle;
  - described by probabilities of events;
- Continuous processes:
  - decay time of an unstable particle;
  - described by probability density function (PDF);



Simulation of natural (stochastic) laws = reproduction of probability distributions.

Generation of samples that follow probability distributions (f(x)) means throwing (many) random numbers.

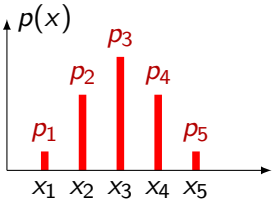
A sequence of random numbers is a set of numbers that have nothing to do with the other numbers in the sequence.

In computer programs we use pseudo-random numbers.
Linear congruential generator:

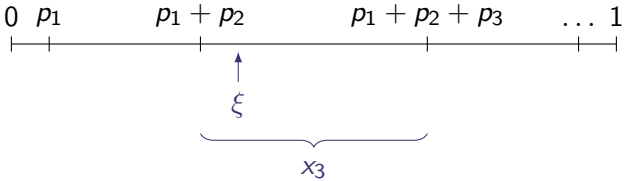
$$I_{n+1} = (aI_n + c) \mod m$$

e.g. $m = 2^{31}$, $a = 1103515245$, $c = 12345$ (for glibc (gcc) implementation)

Discrete sampling

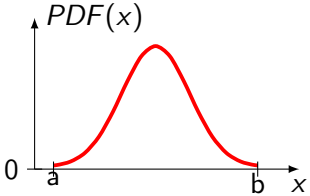


$$\sum_{i=1}^{5} p_i = 1$$

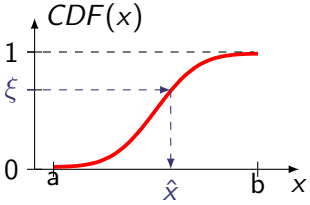


1. Split $[0, 1]$ into intervals corresponding to discrete probabilities p_i.
2. Generate a random number $\xi \in [0, 1]$.
3. Look in which interval it falls.

Continuous sampling - direct method



$$\int_a^b p(x)\dot{x} = 1$$

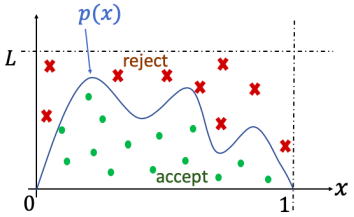


$$P(x) = \int_a^x p(x)\dot{x}$$

1. For probability density function $p(x)$ find the cumulative density function $P(x)$.
2. Generate a random number $\xi \in [0, 1]$.
3. Find $x = \hat{x}$ for which $P(x) = \xi$.

$$\hat{x} = P^{-1}(\xi)$$

Continuous sampling - accept–reject method



M. Loem, Towards Data Science

1. Generate two random numbers $\xi_x \in [0, 1]$ and $\xi_y \in [0, 1]$.

2. Scale them if necessary: $x_i = \xi_x$, $y_i = L\xi_y$.

3. If $y_i > p(x_i) \Rightarrow$ reject x_i,
   If $y_i \leqslant p(x_i) \Rightarrow$ accept x_i.

4. Fraction of accepted points is equal to fraction of area below curve $p(x)$.

Simple example: Particle decay in flight

- Spontaneous process of an unstable particle.
- Decay time is a random value with probability density function

$$f(t) = \frac{1}{\tau} \exp\left(-\frac{t}{\tau}\right), t \geqslant 0$$

τ is the mean life of particle

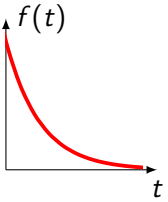
- Probability that particle decays before time T is given by CDF

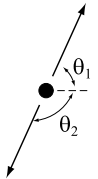
$$F(t) = 1 - \exp\left(-\frac{t}{\tau}\right)$$

which can be used to sample directly.

$$\xi_1 \in [0, 1] \rightarrow \hat{t} = \tau \ln(1 - \xi_1)$$

- Random angles are chosen for decay products, e.g. θ_1 and θ_2 (in rest frame).
- Decay products are boosted to lab frame.





JabberWok, Wikipedia

JabberWok, Wikipedia

Monte Carlo methods

- Monte Carlo name coined by Ulam and Metropolis in 1949 (Manhattan project).
- Recognition of newly invented computer power to application of statistical sampling to solve .
- Metropolis (1948): First actual Monte Carlo calculations using a computer (ENIAC).
- Berger (1963): First complete coupled electron-photon transport code that became known as ETRAN.
- Exponential growth since the 1980's with the availability of computers.

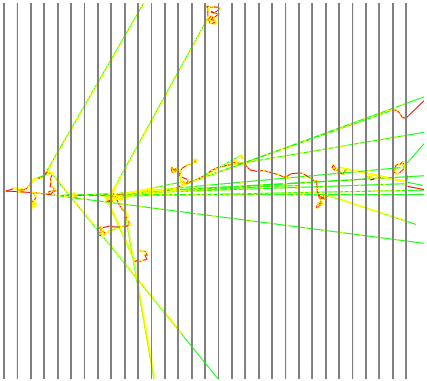
Monte Carlo codes

Non-exhaustive list of Monte Carlo codes

- EM physics
  - ETRAN (Berger & Seltzer; NIST)
  - EGS4 (Nelson, Hirayama, Rogers; SLAC)
  - EGS5 (Hirayama et al.; KEK/SLAC)
  - EGSnrc (Kawrakow & Rogers; NRCC)
  - Penelope (Salvat et al.; U. Barcelona)
- Hadronic physics / general purpose
  - Fluka (Ferrari et al., CERN/INFN)
  - **Geant4** (Geant4 Collaboration)
  - MARS (James & Mokhov; FNAL)
  - MCNPX / MCNP5 (LANL)
  - PHITS (Niita et al.; JAEA)

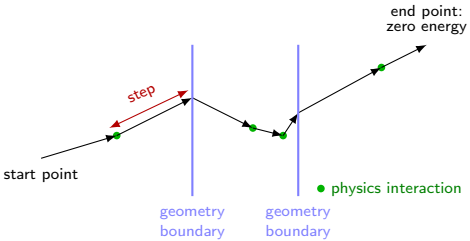
Geant4 is a toolkit used to simulate particle passage through matter.

- Non-deterministic:
  - no equations to be solved,
  - use of random numbers to reproduce distributions.
- General code:
  - allows to describe different geometries (shapes, materials),
  - contains distributions describing various physics processes.
- Finds application in many areas:
  - high energy physics,
  - astrophysics,
  - medical physics,
  - industry.
- Toolkit:
  - no `main` program, set of tools that allows to build user applications.



How does simulation work?

- Track one particle at a time.
- Consider particle passage in steps.
- For each step:
  - Determine **step length** and **interaction** (if any)from cross-sections (probabilities) of physics processes, and geometrical boundaries.
  - **Deposit** energy.
  - If physics process creates **new particles**, add them to the list.
  - **Move** particle to new position, taking into account electromagnetic field.
  - If particle has energy $E > 0$ and is still within detector ("world"), **repeat**. Otherwise, take new particle.



Geant4 application

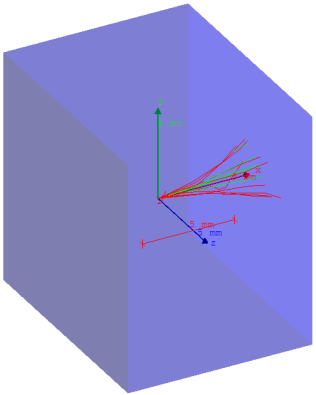
Geant4 is a toolkit = user builds a C++ application, using Geant4 classes.

Basic information on Geant4 is presented, find more in documentation.

There are few components that are mandatory to use, and many more that are optional and depend on user needs.

We will focus on a simple application to understand all the mandatory components.

At the end, we will get results of simulation and compare it to experimental data (for this we'll add few optional components).



Source code of our tutorial application

Application adapted from Getting Started with Geant4 course by Mihaly Novak

Installation

Application requires Geant4 installation, with Qt if visualization is used.

How to install and use Geant4, useful links:

- Installation Guide - how to install, on Linux, MacOS, and Windows.
- Application Developers Guide - how to build an application.
- Geant4 forum - for help on all aspects.

Finally, gnuplot is used to plot the results (comparison of experimental and simulation data).

Key elements

Key elements of Geant4 application:

1. How detector (medium) looks like;
2. What physics processes should be taken into account;
3. What particle(s) enter this detector;
4. What is the result of simulation (optional, it is not mandatory but why do we run simulation if not for results?);

Our application defines those elements in following classes:

Mandatory:

- YourDetectorConstruction
- YourPhysicsList
- YourPrimaryGeneratorAction
- YourActionInitialization

Additional:

- YourDetectorMessenger
- YourEventAction
- YourRunAction
- YourSteppingAction

0. Main

G4RunManager

`G4RunManager` object:

- It's the only **mandatory** manager object that user needs to create. All others (`G4EventManager`, `G4SteppingManger`, etc.) are created and deleted automatically.
- It's role is to **control the flow of a run**, the top level simulation unit
- It sets the initialisation of the run i.e. geometry building, setting up the simulation environment
- All information needs to be given to the `G4RunManager` by the user through the interfaces provided by the Geant4 toolkit (we will see them one by one):
  - `G4VUserDetectorConstruction` **(mandatory)**: how the geometry should be constructed (built)
  - `G4VUserPhyscsList` **(mandatory)**: all the particles and their physics interactions to be simulated
  - `G4VUserActionInitialization` **(mandatory)**:
    - `G4VUserPrimaryGeneratorAction` **(mandatory)**: how the primary particle(s) in an event should be produced
    - additional, optional user actions (`G4UserRunAction`, `G4UserEventAction`, `G4UserSteppingAction`, etc..)

Note:
`G4RunManager` for Geant4$\geqslant$10.7 can be created with type `G4RunManagerType::Serial` for serial execution, `G4RunManagerType::MT` for multi-threaded, `G4RunManagerType::Tasking` for tasking mechanism.

Main: yourMainApplication.cc

```
22  int main(int argc, char** argv) {

44      // Construct the default run manager
45      auto runManager =
46        G4RunManagerFactory::CreateRunManager(G4RunManagerType::Serial);

49      // Set (MANDATORY) User initialization classes:
50      //  = 1. G4VUserDetectorConstruction
51      //  = 2. G4VUserPhysicsList
52      //  = 3. G4VUserActionInitialization (that contains G4VUserPrimaryGeneratorAction)

55      YourDetectorConstruction* detector = new YourDetectorConstruction;
56      runManager->SetUserInitialization( detector );

60      const G4String plName = "FTFP_BERT_EMZ";
61      G4PhysListFactory plFactory;
62      G4VModularPhysicsList *pl = plFactory.GetReferencePhysList( plName );
63      runManager->SetUserInitialization( pl );

72      runManager->SetUserInitialization( new YourActionInitialization( detector ) );

105     delete runManager;
106   return 0;
107 }
```

1. Detector geometry

Detector geometry

Material:

- made of elements, which are made of isotopes,
- with macroscopic properties (density, state, pressure, ...).

Solid:

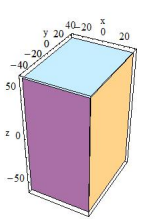
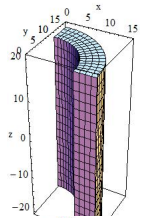
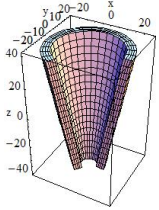
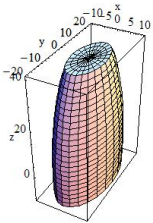
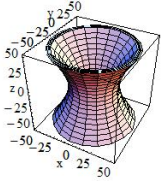
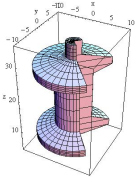
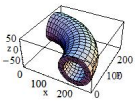
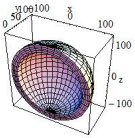
- box, sphere, tube, ...,
- boolean operations on solids.

Logical volumes:

- combine solid with material,
- attach daughter volumes.

Physical volumes:

- placement of logical volume,
- can reuse same logical volume.

Elements and isotopes:

- G4Element object without specifying the isotope composition:

```
// simple way of Carbon element definition
G4Element* elC  = new G4Element(name="Carbon", symbol="C", z = 6., a = 12.01*g/mole);
```

  - need to give: name, symbol, Z and A (effective atomic number and molar mass)
  - isotopes will be automatically added with natural abundances (A won't be updated)

- G4Element object by specific (non-natural) isotope composition:

```
// Define "enriched uranium" element as 90 % of U235 and 10 % of U 238:
//
// create the isotopes: iz = number of protons and n = number of nucleons
G4Isotope* U5 = new G4Isotope(name="U235", iz=92, n=235);
G4Isotope* U8 = new G4Isotope(name="U238", iz=92, n=238);
// create the element and build up by adding the isotopes with their abundance
G4Element* elU=new G4Element(name="enriched uranium",symbol="U",numisotopes=2);
elU->AddIsotope(U5, abundance= 90.*perCent);
elU->AddIsotope(U8, abundance= 10.*perCent);
```

  - element object must be created: name, symbol, number of isotopes
  - isotope objects must be created: name, number of protons and nucleons
  - isotopes need to be added by their relative abundance

Simple G4Material object definition:

- "simple": the material contains only one element and the corresponding G4Element object is not provided:

```
// single element "Uranium" material without giving the uranium element object
G4Material* matU = new G4Material(name    = "Uranium",
                                  z       = 92.0,
                                  a       = 238.03 * g/mole,
                                  denisty= 18.950 * g/cm3);
```

  - the corresponding G4Element object will be automatically created (with natural isotope abundance)
  - need to give: name, density of the material, Z and A (effective atomic number and molar mass) of the single G4Element
  - what happens if we want the single element to have non-natural isotope abundance e.g. the previously created enriched uranium (see later)

Material - molecule

G4Material object definition as chemical molecule:

- G4Material object definition as chemical molecule:
- molecules build up from (several) elements with composition specified by the number of element (e.g. water = H2O)
- accordingly, G4Material object can be created by adding G4Element objects to it together with their composition number:

```
// Create water material as molecule based on its chemical formula (H2O)
//
// create the necessary H and O elements (natural isotope abundance):
G4Element* elH  = new G4Element(name   = "Hydrogen",
                                symbol = "H",
                                z      =  1.00,
                                a      =  1.01 * g/mole);
G4Element* elO  = new G4Element(name   = "Oxygen",
                                symbol = "O",
                                z      =  8.00,
                                a      = 16.00 * g/mole);
// create the water material (name, density, number of components):
G4Material* matH2O = new G4Material(name        = "Water",
                                    density     = 1.0 * g/cm3,
                                    ncomponents = 2);
```

Material - mixture

G4Material object definition as mixture:

- mixture of elements (G4Element), mixture of other materials (G4Material) or even mixture of elements and materials
- similar to molecules with the differences: components can be other materials not only elements the ratio of the components must be given as "fractional mass" not as "number of atoms"
- mixture of elements example: using the AddElement method

```
// Create air material as 70-30 % mixture of N and O elements
// (assuming that N and O Geant4 elements have already been created as
//  elN and elO)
//
// create the air material (name, density, number of components):
G4Material* matAir = new G4Material(name        = "Air",
                                    density     =  1.290 * mg/cm2,
                                    ncomponents =  2);
// add the elements to the material with their fractional mass
matAir->AddElement(elN, fractionmass = 0.7);
matAir->AddElement(elO, fractionmass = 0.3);
```

Material - NIST database

- The data base includes more than 3000 isotopes
- Isotopic composition of elements (Z = [1-108]) with their natural isotopic abundance: using the NIST Atomic Weights and Isotopic Compositions data base
- NIST elements can be obtained easily from the Geant4 NIST data base by using their symbol or Z - atomic number:
  - the corresponding G4Isotope objects will be automatically built
  - "find or build" i.e. avoids duplication of element objects

```
// get the carbon G4Element object from the NIST data base: by its symbol
G4Element* elC  = G4NistManager::Instance()->FindOrBuildElement("C");
// get the silicon G4Element object from the NIST data base: by its Z
G4Element* elSi = G4NistManager::Instance()->FindOrBuildElement(14);
```

- Large collection of pre-defined materials:
  - pre-defined: density, elemental composition (with the pre-defined natural isotopic composition), mean ionization energy, density effect parameters, etc.

Material - NIST database

- Use these pre-defined materials whenever possible:
  - guaranties high accuracy for many derived parameters (consistency)
  - full list of defined materials
- NIST and more predefined materials (318 at the moment):
  - single element NIST materials with $Z = [1 - 98]$ and named after the atomic symbol:
    aluminum (`G4_Al`), silicon (`G4_Si`), gold (`G4_Au`), etc.
  - compound NIST materials:
    `G4_AIR`, `G4_ALUMINUM_OXIDE`, `G4_MUSCLE_SKELETAL_ICRP`, etc.
  - HEP and nuclear materials:
    liquid argon `G4_lAr`, lead tungstate `G4_PbWO4`, `G4_STAINLESS-STEEL`, etc.
  - space materials:
    `G4_KEVLAR`, `G4_NEOPRENE`, etc.
  - bio-chemical materials:
    the DNA bases `G4_ADENINE`, `G4_GUANINE`, `G4_CYTOSINE`, `G4_THYMINE`, etc.

Z		A	m	error	(%)	A_{eff}
14	Si	22	22.03453	(22)		28.0855(3)
		23	23.02552	(21)		
		24	24.011546	(21)		
		25	25.004107	(11)		
		26	25.992330	(3)		
		27	26.98670476	(17)		
		28	27.9769265327	(20)	92.2297 (7)	
		29	28.97649472	(3)	4.6832 (5)	
		30	29.97377022	(5)	3.0872 (5)	
		31	30.97536327	(7)		
		32	31.9741481	(23)		
		33	32.978001	(17)		
		34	33.978576	(15)		
		35	34.984580	(40)		
		36	35.98669	(11)		
		37	36.99300	(13)		
		38	37.99598	(29)		
		39	39.00230	(43)		
		40	40.00580	(54)		
		41	41.01270	(64)		
		42	42.01610	(75)		

Material - NIST database

- Use these pre-defined materials whenever possible:
  - can be obtained from the Geant4 NIST data base by using their name
  - their name starts with the G4_ prefix (see previous slide)
  - full list of defined materials can be found in Developers Guide

```
// Use the NIST data base to get predefined materials: carbon, silicon
//
// get the simple pre-defined carbon material from the NIST data base
G4Material* matC  = G4NistManager::Instance()->FindOrBuildMaterial("G4_C");
// get the simple pre-defined silicon material from the NIST data base
G4Material* matSi = G4NistManager::Instance()->FindOrBuildMaterial("G4_Si");
```

```
// Use the NIST data base to get pre-defined materials:
//
// get the NIST manager (just to simplify)
G4NISTManager* nistMGR = G4NistManager::Instance();
// get the pre-defined liquid argon ("G4_lAr") from the NIST DB
G4Material* matLAr     = nistMGR->FindOrBuildMaterial("G4_lAr");
// get the pre-defined concerete ("G4_CONCRETE") from the NIST DB
G4Material* matConcr   = nistMGR->FindOrBuildMaterial("G4_CONCRETE");
```

Detector geometry, materials: src/YourDetectorConstruction.cc

```
20 YourDetectorConstruction::YourDetectorConstruction()
21 :   G4VUserDetectorConstruction(),
24     // set default target material to be Silicon
25     SetTargetMaterial("G4_Si");
31 }
39 void YourDetectorConstruction::SetTargetMaterial(const G4String& matName) {
40     // try to find the material in the NIST DB
41     fTargetMaterial = G4NistManager::Instance()->FindOrBuildMaterial(matName);
49 }
55 G4VPhysicalVolume* YourDetectorConstruction::Construct() {
56   // I. CREATE MATERIALS:
57     // (note that we use fixed material here one could use messenger to set them)
58     // 1. Material for the world: low density hydrogen defined by "hand"
59     G4double zet      = 1.0;
60     G4double amass    = 1.01*CLHEP::g/CLHEP::mole;
61     G4double density  = CLHEP::universe_mean_density;
62     G4double pressure = 3.e-18*CLHEP::pascal;
63     G4double tempture = 2.73*CLHEP::kelvin;
64     G4Material* materialWorld  = new G4Material("Galactic", zet, amass, density,
65                                                 kStateGas, tempture, pressure);
66     // 2. Material for the target: material pointer stored in fTargetMaterial
67     G4Material* materialTarget = fTargetMaterial;
118 }
```

Geometry description

Geant4 detector geometry description is composed of three conceptual layers:

- solid,
- logical volume,
- physical volume.
- Users need to construct all those directly in their user code (Detector Construction) by `new`, they get registered at construction in the corresponding store (`G4SolidStore`, `G4LogicalVolumeStore`, `G4PhysicalVolumeStore`) which will take care of deallocation of the corresponding memory at the end (if needed).
- More information on the detector geometry description can be found in the corresponding documentation.

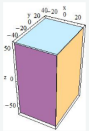
Solid: Box

`G4VSolid`:

- the shape of the Geant4 detector geometry builds up from geometrical primitives, all derived from the `G4VSolid` base class that provides interface to:
  - compute distances between the shape and a given point
  - check whether a point is inside the shape
  - compute the extent of the shape
  - compute the surface normal to the shape at a given point
- More information in documentation.

To create a **box** one can use the constructor:

```
G4Box(const G4String& pName,
            G4double  pX,
            G4double  pY,
            G4double  pZ)
```



In the picture:

`pX = 30, pY = 40, pZ = 60`

by giving the box a name and its half-lengths along the X, Y and Z axis:

`pX`	half length in X	`pY`	half length in Y	`pZ`	half length in Z

This will create a box that extends from `-pX` to `+pX` in X, from `-pY` to `+pY` in Y, and from `-pZ` to `+pZ` in Z.

For example to create a box that is 2 by 6 by 10 centimeters in full length, and called `BoxA` one should use the following code:

```
G4Box* aBox = new G4Box("BoxA", 1.0*cm, 3.0*cm, 5.0*cm);
```

Logical volume

`G4LogicalVolume`:

- encapsulates all information of a detector volume element except its real physical position (position and rotation):
  - the shape and dimensions of the volume i.e. a G4VSolid
  - the material of the volume i.e. G4Material that is the minimally required additional information beyond the solid
  - additional, optional information such as magnetic field (`G4FieldManager`) or user defined limits (`G4UserLimits`), etc.
- More information in documentation.

```
G4LogicalVolume( G4VSolid*              pSolid,         // Its solid
                 G4Material*            pMaterial,      // Its material
                 const G4String&        Name,           // Its name
                 G4FieldManager*        pFieldMgr=0,
                 G4VSensitiveDetector*  pSDetector=0,
                 G4UserLimits*          pULimits=0,
                 G4bool                 Optimise=true )
```

Physical volume

`G4VPhysicalVolume`:

- the abstract base class for representation of physically positioned volumes
- a volume is positioned in a mother volume relative to its coordinate system
- the positioning can be:
  - **placement volume:** one positioned volume, i.e. one `G4VPhysicalVolume` object represents one 'real" volume
  - **repeated volume:** one volume positioned many times, i.e. one `G4VPhysicalVolume` object represents multiple copies of "real" volumes (reduces memory by exploiting symmetry)
    - Replica volumes: the multiple copies of the volume are all identical
    - Parameterised volumes: the multiple copies of a volume can be different in size, solid type, or material that can all be parameterised as a function of the copy number
- Here focus on **placement volume**, more information in documentation.

Physical volume

`G4PVPlacement`:

- represent one positioned `G4LogicalVolume`
- created by associating a `G4LogicalVolume` with a transformation that defines the position of the volume in the mother volume
- the transformation can be given either as a single G4Transform3d object or as combination of rotation `G4RotationMatrix` and translation `G4ThreeVector`
- a mother volume must be specified for all volumes except the "world"
- (one of the two) constructor with the rotation matrix and translation vector:

```
G4PVPlacement(  G4RotationMatrix*    pRot,  // Rotation with respect to its mother volume
                const G4ThreeVector& tlate, // Translation with respect to its mother vol
                G4LogicalVolume*     pCurrentLogical, // Its logical Volume
                const G4String&      pName,           // Its name
                G4LogicalVolume*     pMotherLogical,  // Its mother volume
                G4bool               pMany,           // Can be set to false, not used
                G4int                pCopyNo,         // Its identifier
                G4bool               pSurfChk=false ) // Check for overlaps
```

G4VUserDetectorConstruction

`G4VUserDetectorConstruction` interface is provided by the Geant4 toolkit to describe the geometrical setup, including all volumes with their shape, position and material definition

- Its `G4VUserDetectorConstruction::Construct()` interface method (pure virtual) is invoked by the `G4RunManager` at initialisation.
- It's **mandatory** to implement a derived class (`YourDetectorConstruction` in our example) from this base class.
  - create all materials will need to use in your geometry
  - describe your detector geometry by creating and positioning all volumes
  - return the pointer to the root of your geometry hierarchy i.e. the pointer to your "World" `G4VPhysicalVolume`
- User object `YourDetectorConstruction` needs to be created and registered in `G4RunManager` object by using the `G4RunManager:SetUserinitialization` method.

Note: For non-serial execution of Geant4 (multithreaded, tasking), `Construct()` interface method is invoked only by the master thread in case (i.e. only one detector object), while the other `ConstructSDandField()` interface method is invoked by each worker threads (i.e. thread local objects created).

Detector geometry, dimensions: src/YourDetectorConstruction.cc

```
20 YourDetectorConstruction::YourDetectorConstruction()
21 :   G4VUserDetectorConstruction(),

26     // set default thickness
27     fTargetThickness = 1.0*CLHEP::cm;

31 }

55 G4VPhysicalVolume* YourDetectorConstruction::Construct() {

69     // II. CREATE GEOMETRY:
70     // 1. Define target and world sizes
71     G4double targetXSize  = fTargetThickness;
72     G4double targetYZSize = 1.25*targetXSize;
73     G4double worldXSize   = 1.1*targetXSize;
74     G4double worldYZSize  = 1.1*targetYZSize;

118 }
```

Detector geometry, world: src/YourDetectorConstruction.cc

```
55 G4VPhysicalVolume* YourDetectorConstruction::Construct() {

76     // 2. Create the world and the target (both will be box):
77     // a. world
78     G4Box*                worldSolid
79       = new G4Box("solid-World",  // name
80       0.5*worldXSize,   // half x-size
81       0.5*worldYZSize,  // half y-size
82       0.5*worldYZSize); // half z-size
83     G4LogicalVolume*    worldLogical
84       = new G4LogicalVolume(worldSolid,      // solid
85                             materialWorld,  // material
86                             "logic-World"); // name
87     G4VPhysicalVolume*  worldPhysical
88       = new G4PVPlacement(nullptr,                    // (no) rotation
89                           G4ThreeVector(0.,0.,0.), // translation
90                           worldLogical,            // its logical volume
91                           "World",                 // its name
92                           nullptr,                 // its mother volume
93                           false,                   // not used
94                           0);                      // cpy number

118 }
```

Detector geometry, target: src/YourDetectorConstruction.cc

```
55 G4VPhysicalVolume* YourDetectorConstruction::Construct() {

95     // b. target
96     G4Box*              targetSolid
97       = new G4Box("solid-Target",    // name
98                   0.5*targetXSize,   // half x-size
99                   0.5*targetYZSize,  // half y-size
100                  0.5*targetYZSize); // half z-size
101    G4LogicalVolume*    targetLogical
102      = new G4LogicalVolume(targetSolid,    // solid
103                            materialTarget,  // material
104                            "logic-Target"); // name
105    G4VPhysicalVolume*  targetPhysical
106      = new G4PVPlacement(nullptr,                    // (no) rotation
107                          G4ThreeVector(0.,0.,0.), // translation
108                          targetLogical,           // its logical volume
109                          "Target",                // its name
110                          worldLogical,            // its mother volume
111                          false,                   // not used
112                          0);                      // cpy number

118 }
```

2. Physics interactions

Main electromagnetic processes

Gamma

- Conversion
  γ -> e+ e- , μ+ μ-
- Compton scattering
  γ *(atomic)*e- ---> γ *(free)*e-
- Photo-electric
  γ material ---> *(free)*e-
- Rayleigh scattering
  γ atom ---> γ atom

Muon

- Pair production
  μ- atom ---> μ- e+ e-
- Bremsstrahlung
  μ- (atom) ---> μ- γ
- MSC (Coulomb scattering) :
  μ- atom ---> μ- atom
- Ionization
  μ- atom ---> μ- ion+ e-

Total cross section :
⇨ step length

Differential & partial cross sections :
⇨ final state (multiplicity & spectra)

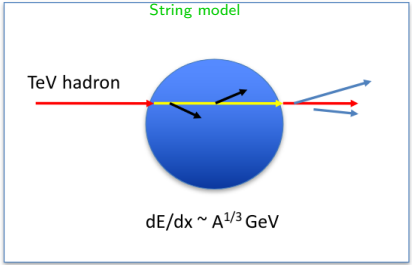
Electron, Positron

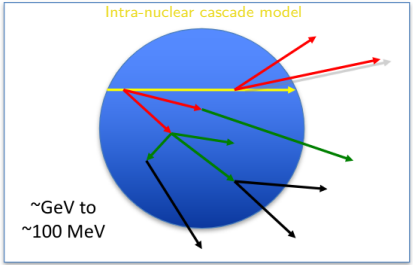
- Bremsstrahlung
  e- (atom) ---> e- γ
- MSC (Coulomb scattering)
  e- atom ---> e- atom
- Ionization
  e- atom ---> e- ion+ e-
- Positron annihilation
  e+ e- ---> γ γ

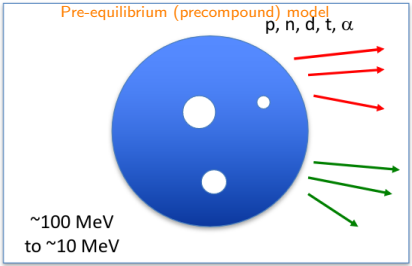
Charged hadron, ion

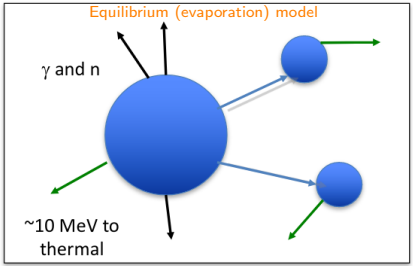
- Bremsstrahlung
  h- (atom) ---> h- γ
- MSC (Coulomb scattering)
  h- atom ---> h- atom
- Ionization
  h- atom ---> h- ion+ e-

Hadronic interactions from TeV to MeV



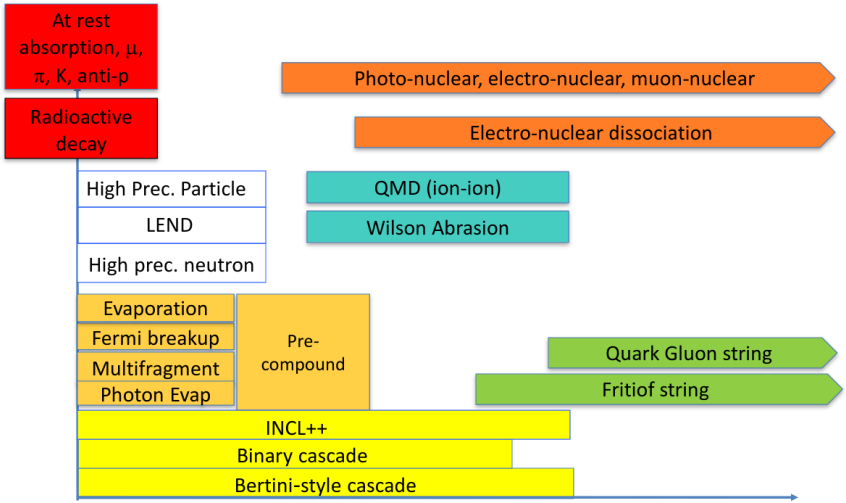






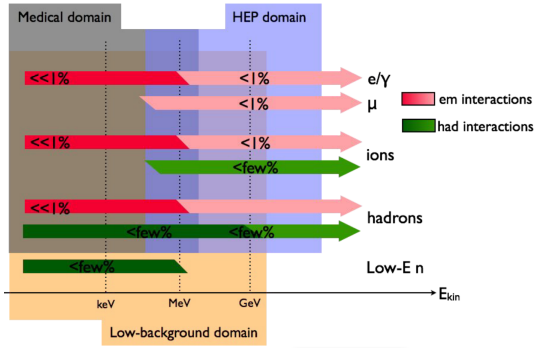
D. Wright, Hadronic Physics

Partial hadronic model inventory



D. Wright, Hadronic Physics

Physics list



A. Dotti, Detector Simulations

- Geant4 is a toolkit used in various areas, with different needs for precision, speed, models.
- No single "physics" that fits everyone needs.
- Atomistic approach — provide independent (and alternative) processes.
- Combine particle definitions and processes into physics lists.

G4VUserPhysicsList

`G4VUserPhysicsList` interface is provided by the Geant4 toolkit to describe the physics setup, including definition of all particles and their physics interactions, processes.

- Its `G4VUserPhysicsList::ConstructParticle()` and `::ConstructProcess()` interface methods (pure virtual) are invoked by the `G4RunManager` (actually by the `G4RunManagerKernel` and process construction is invoked indirectly) at initialisation
- It's **mandatory** to use a derived class from this base class with implementation of the particles in `ConstructParticle()` and processes in `ConstructProcess()` methods.
- Manual (user) construction of a physics list is recommended only for advanced users!
- Geant4 provides possibilities with different level of granularity to build up or obtain a complete pre-defined physics list.
- Physics list needs to be created and registered in `G4RunManager` object by using the `G4RunManager:SetUserinitialization` method. It's just few lines of code!

Physics list: yourMainApplication.cc

Modular physics lists are ready-to-use, complete lists with particles, processes, and transport defined.

Geant4 can only recommend certain physics list for usual use cases, it's up to user to validate them for custom application.

Documentation of physics lists.

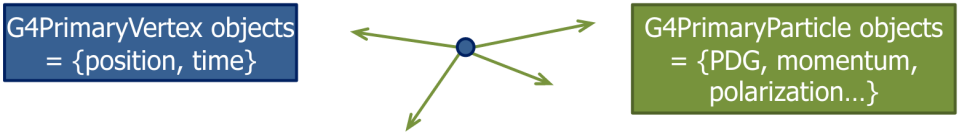
```
60      const G4String plName = "FTFP_BERT_EMZ";
61      G4PhysListFactory plFactory;
62      G4VModularPhysicsList *pl = plFactory.GetReferencePhysList( plName );
63      runManager->SetUserInitialization( pl );
```

FTFP_BERT physics list is recommended for high energy physics.
EMZ is collection of most accurate electromagnetic models.

3. Primary particles

Primary particles

- **Primary particle(s)** means particle(s) with which you **start an event**.
  - E.g. particles made by the primary p-p collision, an alpha particle emitted from radioactive material, a gamma-ray from treatment head, etc.
  - Then Geant4 tracks these primary particles in your geometry with physics interactions and generates secondaries, detector responses and/or scores.
- **Primary vertex** has **position** and **time**. Primary particle has a particle ID, momentum and optionally polarization. One or more primary particles may be associated with a primary vertex. One event may have one or more primary vertices.



- Generation of primary vertex/particle is one of the user-mandatory tasks. G4VUserPrimaryGeneratorAction is the abstract base class to control the generation.
  - Actual generation should be delegated to G4VPrimaryGenerator class. Several concrete implementations, e.g. G4ParticleGun, G4GeneralParticleSource, are provided.

Particle gun vs. General Particle Source

Particle Gun
`G4ParticleGun`

- Simple and naïve
- Shoot one track at a time
- Easy to handle.
  - Use set methods to alternate track–by–track or event–by–event values.

General Particle Source
`G4GeneralParticleSource`

- Powerful
- Controlled by UI commands.
  - Almost impossible to control through set methods
- Capability of shooting particles from a surface of a volume.
- Capability of randomizing kinetic energy, position and/or direction following a user- specified distribution (histogram).

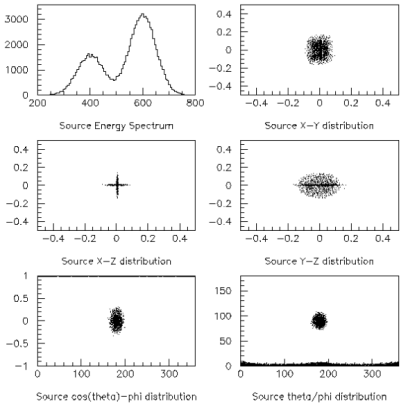
- If you need to shoot primary particles from a surface of a volume, either outward or inward, GPS is the choice.
- If you need a complicated distribution, not flat or Gaussian, GPS is the choice.
- Otherwise, use Particle Gun.

General Particle Source

```
# two beams in a generator
#
# beam #1
# default intensity is 1 now change to 5.
/gps/source/intensity 5.
#
/gps/particle proton
/gps/pos/type Beam
#
# the incident surface is in the y-z plane
/gps/pos/rot1 0 1 0
/gps/pos/rot2 0 0 1
#
# the beam spot is centered at the origin and is of
# 1d gaussian shape with a 1 mm central plateau
/gps/pos/shape Circle
/gps/pos/centre  0. 0. 0. mm
/gps/pos/radius 1. mm
/gps/pos/sigma_r .2 mm
#
# the beam is travelling along the X_axis with
# 5 degrees dispersion
/gps/ang/rot1 0 0 1
/gps/ang/rot2 0 1 0
/gps/ang/type beam1d
/gps/ang/sigma_r 5. deg
#
# the beam energy is in gaussian profile
# centered at 400 MeV
/gps/ene/type Gauss
/gps/ene/mono 400 MeV
/gps/ene/sigma 50. MeV
```

(macro continuation...)

```
# beam #2
# 2x the instensity of beam #1
/gps/source/add 10.
#
# this is a electron beam
/gps/particle e-
/gps/pos/type Beam
# it beam spot is of 2d gaussian profile
# with a 1x2 mm2 central plateau
# it is in the x-y plane centred at the orgin
/gps/pos/centre  0. 0. 0. mm
/gps/pos/halfx 0.5 mm
/gps/pos/halfy 1. mm
/gps/pos/sigma_x 0.1 mm
# the spread in y direction is stronger
/gps/pos/sigma_y 0.2 mm
#
#the beam is travelling along -Z_axis
/gps/ang/type beam2d
/gps/ang/sigma_x 2. deg
/gps/ang/sigma_y 1. deg
# gaussian energy profile
/gps/ene/type Gauss
/gps/ene/mono 600 MeV
/gps/ene/sigma 50. MeV
```



Documentation of GPS

G4VUserPrimaryGeneratorAction

`G4VUserPrimaryGeneratorAction` interface is provided by the Geant4 toolkit to describe how the primary particle(s) in an event should be produced

- `G4VUserPrimaryGeneratorAction::GeneratePrimaries()` interface method (pure virtual) is invoked by the `G4RunManager` during the event-loop (in its `G4RunManager::GenerateEvent()` method).
- It's **mandatory** to implement a derived class (`YourPrimaryGeneratorAction` in our example) from this base class.
  - describe how the primary particle(s) in an event should be produced
  - we will use a `G4ParticleGun` object, provided by the Geant4 toolkit, to generate primary particles: one particle per event with defined kinematics
- User object `YourPrimaryGeneratorAction` needs to be created within user implementation of `G4VUserActionInitialization::Build()` method, e.g. `YourActionInitialization::Build()`.

Note:

- the Detector-Construction and the Physics-List need to be created directly in the main program and registered directly in the `G4RunManager` object
- all User-Actions needs to be created and registered in the User-Action-Initialisation (including the only mandatory Primary-Generator-Action as well as all other, optional User-Actions)

G4VUserActionInitialization

`G4VUserActionInitialization` interface is provided by the Geant4 toolkit to create and register user actions.

- That includes:
  - the **only one mandatory** `G4VUserPrimaryGeneratorAction` user action.
  - all other optional user actions (`G4UserRunAction`, `G4UserEventAction`, etc..).
- Its `G4VUserActionInitialization::Build()` interface method (pure virtual) is invoked by the `G4RunManager` at initialisation
- It's **mandatory** to implement a derived class (`YourActionInitialization`) from this base class.
  - Primary particle generator (object of `YourPrimaryGeneratorAction` class) needs to be created in `YourActionInitialization::Build()` method.
  - All optional user actions may also be created.
- User object `YourActionInitialization` needs to be created and registered in `G4RunManager` object by using the `G4RunManager:SetUserinitialization` method.

Note:

- For non-serial execution, `Build()` method is executed by every worker thread, while `BuildForMaster()` method is invoked only by the master thread.
- The only user action that is supposed to be created in `BuildForMaster()` method, is the implementation of the `G4UserRunAction` for the master thread (e.g. to create and merge the output of simulation).

Actions initialization: src/YourActionInitialization.cc

```
13
14 YourActionInitialization::YourActionInitialization(YourDetectorConstruction* det)
15 :   G4VUserActionInitialization(),
38
39 // Create all User Actions here:
40 // - for sequential mode (will be invoked immediately by the only one G4RunManager
41 //   when the ActionInitialization object is registered in it in the main)
42 // - for worker threads (will be invoked later by all worker G4RunManager-s)
43 void YourActionInitialization::Build() const {
44   // Set UserPrimaryGeneratorAction
45   YourPrimaryGeneratorAction* primaryAction = new YourPrimaryGeneratorAction(fYourDetector);
54   SetUserAction( new YourSteppingAction(fYourDetector, eventAction) );
```

DetectorConstruction pointer is propagated down to user actions because primary generator needs information from the detector (it defines where particles are created, so they enter the detector).

Primary partcles: src/YourPrimaryGeneratorAction.cc

```
 9
10 YourPrimaryGeneratorAction::YourPrimaryGeneratorAction(YourDetectorConstruction* det)
11 :   G4VUserPrimaryGeneratorAction(),
12     fYourDetector(det),
13     fParticleGun(nullptr) {
14     // create the particle-gun object
15     G4int nParticle = 1;
16     fParticleGun    = new G4ParticleGun(nParticle);
17     SetDefaultKinematic();
18
19 void YourPrimaryGeneratorAction::GeneratePrimaries(G4Event* evt) {
20   fParticleGun->GeneratePrimaryVertex(evt);
21
22 void YourPrimaryGeneratorAction::SetDefaultKinematic() {
23     //
24     // default primary particle: 30 [MeV] e- perpendicular to the target
25   G4ParticleDefinition* part = G4ParticleTable::GetParticleTable()->FindParticle( "e-" );
26     fParticleGun->SetParticleDefinition( part );
27     fParticleGun->SetParticleMomentumDirection( G4ThreeVector(1., 0., 0.) );
28     fParticleGun->SetParticleEnergy( 30.*CLHEP::MeV );
29     UpdatePosition();
30
31 // needs to be invoked for all workers at the begining of the run: user might
32 // have changed the target thickness
33 void YourPrimaryGeneratorAction::UpdatePosition() {
34     fParticleGun->SetParticlePosition(
35         G4ThreeVector( fYourDetector->GetGunXPosition(), 0.0, 0.0 ) );
```

4. Storing the results

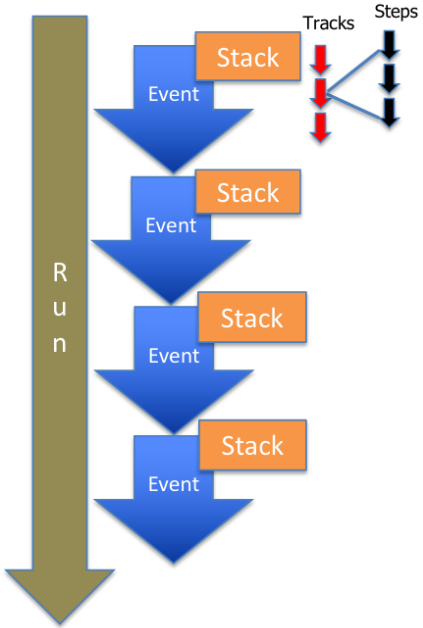
User Actions

Mandatory users actions base classes:

- `G4VUserActionInitialization`
- `G4VUserPrimaryGeneratorAction`

Optional user action base classes:

- Can work on various levels:
  - `G4UserRunAction`
  - `G4UserEventAction`
  - `G4UserTrackingAction`
  - `G4UserSteppingAction`
  - `G4UserStackingAction`
- Fully customizable (empty by default)
  - provide `BeginOf...` and `EndOf...` methods called automatically by the kernel



Simulation results

All mandatory components allow to implement a working simulation.

However, there is no output of the simulation (no measurement).

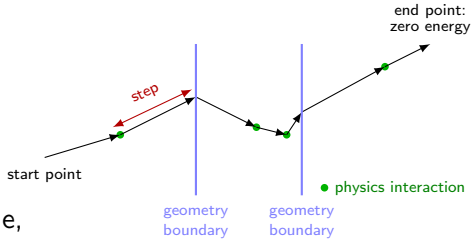
In experiments, particle passage through detector can be detected thanks to its interactions in **certain** regions.

This must be implemented in our application: in which regions, what particles, how is it stored.

Different ways of implementation are illustrated in basic example B4 of Geant4.

Our example application uses purely user actions. Different methods include use of sensitive detector or scorers.

Simulation results: user actions



For each step:

- check if it is located within our target volume,
- if yes: get value of the energy deposited in medium,
  and add it to per-event mean calculation.

For each event:

- accumulate energy deposited in single steps within target,
- at the end of event processing add entry to a histogram.

For each run:

- create an output file and a histogram to store energy deposit,
- at the end of run scale the histogram and store it.

Stepping action: src/YourSteppingAction.cc

```
17 // Score only if the step was done in the Target:
18 //  - cllect energy deposit for the mean (per-event) energy deposit computation
19 //  - same for the charged particle track length
20 void YourSteppingAction::UserSteppingAction(const G4Step* theStep) {
21   // Score steps done only in the target: i.e. pre-step point was in target
22   if (theStep->GetPreStepPoint()->GetTouchableHandle()->GetVolume()
23     != fYourDetector->GetTargetPhysicalVolume() )  return;
24   // Step was done inside the Target so do scoring:
25   //
26     // Get the energy deposit
27     const G4double eDep    = theStep->GetTotalEnergyDeposit();
28     // add current energy deposit to the charged particle track length per-event
29     fYourEventAction->AddEnergyDepositPerStep( eDep );
30 }
```

Event action: src/YourEventAction.cc

```
15
16 // Beore each event: reset per-event variables
17 void YourEventAction::BeginOfEventAction(const G4Event* /*anEvent*/) {
18   fEdepPerEvt           = 0.0;
19
20
21
22 // After each event:
23 // fill the histogram with energy deposited in this event
24 void YourEventAction::EndOfEventAction(const G4Event* /*anEvent*/) {
25   // Get analysis manager
26   auto analysisManager = G4AnalysisManager::Instance();
27   // Fill histogram
28   analysisManager->FillH1(0, fEdepPerEvt);
```

Run action: src/YourRunAction.cc

```
13 YourRunAction::YourRunAction(YourDetectorConstruction* det, YourPrimaryGeneratorAction* prim)
14 :   G4UserRunAction() {
15     // Create analysis manager
16     auto analysisManager = G4AnalysisManager::Instance();
17     analysisManager->SetDefaultFileType("csv");
18     analysisManager->SetFileName("Hist_Edep.csv");
19 }
23 void YourRunAction::BeginOfRunAction(const G4Run*) {
24     // Get analysis manager
25     auto analysisManager = G4AnalysisManager::Instance();
26     //
27     // Create histogram to store energy
28     analysisManager->CreateH1("energy", "Energy", 100, 0, 10 * keV);
29     analysisManager->SetH1Ascii(0, true);
30     analysisManager->OpenFile();
31 }
34 void YourRunAction::EndOfRunAction(const G4Run*) {
35   // Scale histogram and write to file
36   auto analysisManager = G4AnalysisManager::Instance();
37   G4double binWidth = analysisManager->GetH1Width(0);
38   G4double factor = 1. / (analysisManager->GetH1(0)->sum_bin_heights() * binWidth);
39   analysisManager->ScaleH1(0, factor);
40   analysisManager->Write();
41   analysisManager->CloseFile();
42 }
```

(optional)
UI commands

User interface

1. Geant4: a toolkit that provides all components.
2. G4 application: C++ application dedicated to specific simulation.
3. End-user: could control simulation w/o C++ knowledge, via user interface (UI) commands.

UI commands are also useful for controlling the application w/o rebuilding it.

UI command syntax

- A UI command (e.g. /run/verbose 1) consists of:
  - command directory
  - command
  - parameter(s)
- A parameter can be a type of string, boolean, integer or double:
  - space is a delimiter
  - use double-quotes ("") for strings
- A parameter can be sometimes omitted. Its default value will be taken in this case:
  - predefined default value or current value according to its definition
  - using the default value for the first parameter while setting the second: /directory/command ! second
    i.e. the exclamation mark ! can be used as a place holder

UI command submission

- Geant4 UI commands can be issued in 3 different ways by:
  - (G)UI interactive command submission
  - batch mode using a macro file
  - hard-coded commands in the application (slow):

```
G4UImanager* UI = G4UImanager::GetUIpointer();
UI->ApplyCommand("/run/verbose 1");
```

- The availability of the individual commands, the ranges of parameters, the available candidates on individual command parameter may vary according to the implementation of your application
- some commands are available only for limited Geant4 application state(s): e.g. `/run/beamOn 100` is available only for Idle states

Macro file

A macro file is an ASCII file that contains UI commands

- All commands must be given with their full-path directories
- Use # for comment a line
  - from the first # to the end of the line will be ignored
  - comment lines will be echoed if `/control/verbose` is set to 2
- Macro file can be executed
  - interactively or in other macro files

```
/control/execute macro_file_name
```

  - hard-coded

```
G4UImanager* UI = G4UImanager::GetUIpointer();
UI->ApplyCommand("/control/execute macro_file_name");
```

Primary particles: built-in UI commands

g4Macro.mac

```
24 # ------------------------------------------------------------------------------
25 # Set the primary generator (i.e. the Particle Gun) properties:
26 # ==============================================================================
27 # set the particle energy to 30 MeV
28 /gun/energy 30 MeV
29 # set the particle type to electron
30 /gun/particle e-
```

List of built-in commands

Detector: custom UI commands

experiment/g4Macro_Meroli_100MeV_electron_5p6um_Si.mac

```
19 # ------------------------------------------------------------------------
20 # Your own UI commands defined by "YourDetectorMessenger" to set some of the
21 # properties (thickness, material) of the target:
22 # ========================================================================
23 # set the target thickness to match the experimental one: 5.6 um
24 /yourApp/det/setTargetThickness 5.6 um
25 # set the target material to match the experimental one: Silicon
26 /yourApp/det/setTargetMaterial G4_Si
```

Documentation of how to define custom commands.

Detector UI commands: src/YourDetectorMessenger.cc

```
11
17      fTargetMaterialCMD(nullptr)
19      //
20      // create the "det" command directory first then add commands
21      fDirCMD = new G4UIdirectory("/yourApp/det/");
23      //
24      // UI command to set the target thickness
25      fTargetThicknessCMD = new
    ↪   G4UIcmdWithADoubleAndUnit("/yourApp/det/setTargetThickness",this);
26      // set the description of the command
27      fTargetThicknessCMD->SetGuidance("Sets the Thickness of the Target.");
28      // name = TagetSizeX; omittable=false i.e. user needs to supply a value
29      fTargetThicknessCMD->SetParameterName("TagetSizeX",false);
30      // set the aceptable range of the parameter value higher than zero
31      fTargetThicknessCMD->SetRange("TagetSizeX>0.");
32      // set the unit category to be length
33      fTargetThicknessCMD->SetUnitCategory("Length");
34      // can be modified at PreInit and Idle state
35      fTargetThicknessCMD->AvailableForStates(G4State_PreInit, G4State_Idle);
36      // in MT mode: do not need to be broadcasted for workers
43  //    fTargetMaterialCMD->SetToBeBroadcasted(false);
53
54  void YourDetectorMessenger::SetNewValue(G4UIcommand* command, G4String newValue)
55  {
56      // set target thickness
57      if (command == fTargetThicknessCMD) {
58          G4double thickness = fTargetThicknessCMD->GetNewDoubleValue(newValue);
59          fYourDetector->SetTargetThickness(thickness);
64      }
```

Our application

Brief description

Mandatory components:

- YourDetectorConstruction:
  - a simple box (shape) as the detector/target filled with with silicon as material,
  - placed in a box (shape) "world" volume filled with low density hydrogen gas.
- YourPhysicsList:
  - we will use one of the pre-defined, ready-to-use physics list provided by the Geant4 toolkit (therefore no need to write any user physics list class like in our case)
- YourPrimaryGeneratorAction:
  - a simple particle gun (G4ParticleGun): generates a single primary particle per event with pre-defined particle type and kinematics pointing toward to our target
- YourActionInitialization:
  - implement the construction and registration of our YourPrimaryGeneratorAction object
- Main method of the application and execute the simulation (YourMainApplication.cc)

Application

```
Geant4TutorialApplication
├── experiment ..... discussed later
├── inc
│   ├── YourActionInitialization.hh
│   ├── YourDetectorConstruction.hh
│   ├── YourDetectorMessenger.hh
│   ├── YourEventAction.hh
│   ├── YourPrimaryGeneratorAction.hh
│   ├── YourRunAction.hh
│   └── YourSteppingAction.hh
├── src
│   ├── YourActionInitialization.cc
│   ├── YourDetectorConstruction.cc
│   ├── YourDetectorMessenger.c
│   ├── YourEventAction.cc
│   ├── YourPrimaryGeneratorAction.cc
│   ├── YourRunAction.cc
│   └── YourSteppingAction.cc
└── ...
```

```
Geant4TutorialApplication
├── ...
├── CMakeLists.txt
├── Readme.txt
├── g4Macro.mac
├── vis.mac
└── yourMainApplication.cc
```

Macros (here `*mac`) contain user interface (UI) commands that allow to change application without re-building it!

Geant4TutorialApplication: how to run it?

1. Assuming Geant4 installation was completed, application can be build:

```
git clone https://gitlab.cern.ch/azaborow/geant4tutorialapplication.git
cd Geant4TutorialApplication/
mkdir build
cd build
cmake ..
make
```

2. To run application in batch mode (no visualization):

```
./yourMainApplication ../g4Macro.mac
```

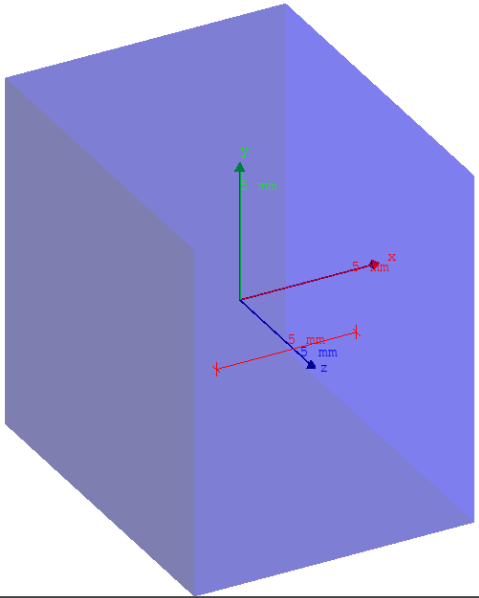
3. With visualization (Qt):

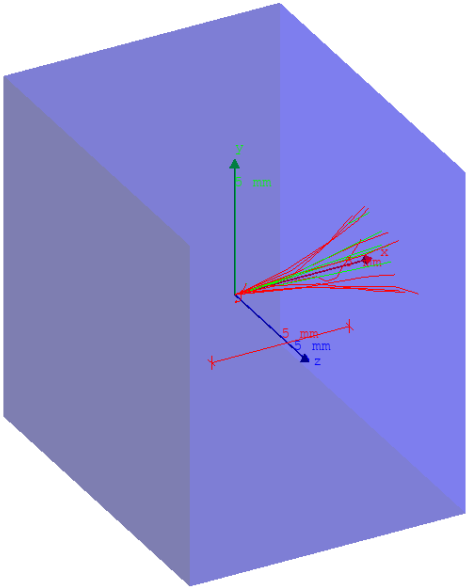
```
./yourMainApplication
```

Then type in the application UI commands:

```
/run/initialize
/control/execute ../vis.mac
/control/execute ../g4Macro.mac
```

Geant4TutorialApplication: visualization





Experiment: electron beam on thin silicon foil

Experimental data is taken from S. Meroli *et al* 2011 JINST 6 P06013:

```
experiment/exp_Meroli_100MeV_electron_5p6um_Si.dat
```

Energy loss distribution of 100 MeV electrons is measured for 5.6 μm thin silicon foil.

To run Geant4 simulation use macro that sets appropriate detector material, thickness, particle type, and its energy.

```
./yourMainApplication ../experiment/g4Macro_Meroli_100MeV_electron_5p6um_Si.mac
```

This produces two files:

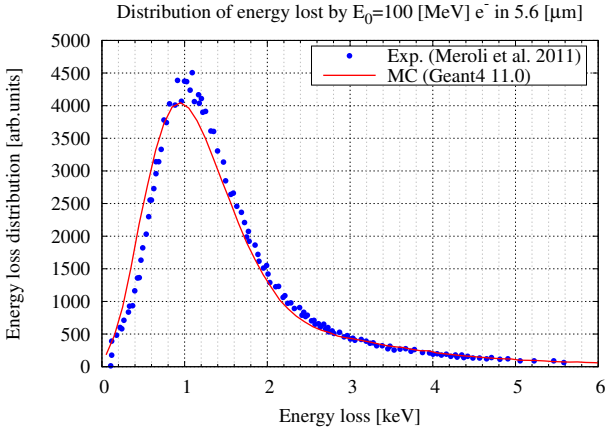
```
Hist_Edep_h1_energy.csv
Hist_Edep.ascii
```

Only second file is used, an example simulation output is copied to `experiment/` directory.

Experiment: electron beam on thin silicon foil

Comparison of simulation to experiment:

```
cd ../experiment
gnuplot gplot.p
xdg-open fig_Meroli_100MeV_electron_5p6um_Si.eps
```



Summary

Summary

We covered most important topics that allow to build a detector simulation using Geant4 toolkit.

Some topics were not included (due to lack of time):

- Visualization: ESIPAP 2021 lecture, Getting Started with Geant4 course, documentation.
- Geant4 examples (distributed with the toolkit): ESIPAP 2021 lecture, documentation.

For any help, anytime, please consult Geant4 documentation and Geant4 forum.

Please contact me on Slack if you have any questions!